

AD-A090 648

MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB
EFFICIENT MULTIPLIERS FOR THE FFT, (U)
JUL 80 S C POHLIG, J M FRANKOVICH

F/6 9/3

UNCLASSIFIED

NL

1 OF 1

ADA
09048



END
DATE
FILMED
11-80
DTIC

AL A090648

Session - 10.

①

EFFICIENT MULTIPLIERS FOR THE FFT*

S. C. Pohlrig and J. M. Frankovich

11 15 July 80

LEVEL II

MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02173
(617) 862-5500

DTIC
ELECTE
301 1 6 1980
S E D

* This work was sponsored by the Department of the Army.

The views and conclusions contained in this document are those of the contractor and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the United States Government.

DIS	MENT A
Ap	release;

80 10 10 010

one file copy

ABSTRACT

One of the major components in a hardware implementation of the discrete Fourier transform (DFT) is the multiplier hardware. There are algorithms available which reduce the number of multiplications used in computing the DFT, and there are hardware techniques for implementing multipliers. This paper presents an efficient method of using table lookup multipliers for the fast Fourier transform (FFT). This method implements the multipliers with a small number of modest size tables.

Accession	
NTIS	X
EDS	
Univ	
Jan	
Letter on File	
By	
Distribution	
Availability Codes	
Dist	Avail and/or special
A	

I. INTRODUCTION

Filtering and spectral analysis are frequently required tasks in signal processing. In digital signal processing, these tasks are often implemented using the Discrete Fourier Transform (DFT), or more specifically the fast Fourier transform (FFT) algorithm.

Radar signal processing provides several applications for the FFT. For example, the FFT can be used in conjunction with a pulse compressor to provide pulse-Doppler processing [1] as shown in Figure 1. In pulse-Doppler processing, the echos of the transmitted pulses are first compressed by a matched filter. Then, for each range cell or time delay, the FFT is applied to the echo of many transmitted pulses. The phase progression across these echos shows up as a Doppler shift in the FFT output.

Another application of the FFT is the implementation of digital filters. It is well known [2] that the convolution of data with a filter can be implemented by taking the FFT of both the data and the filter, multiplying the two results, and taking an inverse FFT. Frequently, this process takes less computation than implementing the filter directly.

The main components used in building FFT hardware are adders, multipliers, and memory. Traditionally, the multipliers have been considered to be the most expensive component in terms of speed, amount of hardware

required, and power consumption. It is not surprising then that much research has been done to develop transform algorithms which reduce the number of multiplications [3,4,5,6,7].

This paper presents several techniques which may be combined in order to implement the multiplications used in the DFT as a table lookup process, thus trading memory for multiplier hardware. Using memories for multipliers in the FFT has been suggested before [8], although the technique is quite different from that presented here.

II. COEFFICIENT FACTORIZATION

The main idea of the technique presented in this paper is to factor the coefficients used in the FFT multipliers in such a way that only a few distinct multiplier coefficients are used. In order to demonstrate this factorization, we first examine multiplication by powers of roots of unity, then apply these results to the FFT.

Let W be an N -th root of unity. It is clear that there are only N different integral powers of W ; W^0, \dots, W^{N-1} . To understand the factorization, suppose that we wish to multiply a data element by W^n for some $n=0, \dots, N-1$. We can write the binary decomposition of n as

$$n = b_{\ell-1}2^{\ell-1} + \dots + b_02^0 \quad (1)$$

where

$$b_i \in \{0, 1\} \quad (2a)$$

and

$$\ell \geq \log_2 N > \ell-1. \quad (2b)$$

Making use of eq. (1) we easily have

$$W^n = \left(W^{2^0}\right)^{b_0} \dots \left(W^{2^{\ell-1}}\right)^{b_{\ell-1}} \quad (3)$$

so that all N powers of W can be formed from the ℓ basic terms

$$W^{2^i}, \quad i=0, \dots, \ell-1 \quad (4)$$

Since ℓ varies with $\log_2 N$, there is a significant reduction in the number of distinct multipliers required to implement multiplication by powers of W as compared with a brute-force approach. This factorization technique is similar to a well know technique for exponentiation [9, p. 398].

The multiplications which occur in the FFT are multiplications by integral powers of an N -th root of unity. Thus, the above discussion applies to multiplications within the FFT. In particular, the decomposition in eq. (3) suggests a particular construction for the multipliers in a FFT hardware realization, as shown in Figure 2. The multiplier is constructed as having ℓ stages, and the data to be multiplied flows from stage to stage in a pipeline fashion. At the i -th stage from the right (for $i=0, \dots, \ell-1$) the data is either multiplied by W^{2^i} (when $b_i=1$) or bypasses the stage (when $b_i=0$).

The next step in the design of the FFT multiplier is to implement the individual stages. The stages corresponding to $i=\ell-2, \ell-1$ have multipliers of $-j$ and -1 , both of which are very simple and require only a small amount of hardware. For the smaller values of i , this implementation may be done as shown in Figure 3. The multiplication by the complex number W^{2^i} is shown as four real multiplications followed by two additions. Another arrangement for the complex multiplier is possible which uses three adders and three real multipliers.

Each of the real multiplications in Figure 3 is multiplication by a constant and can be implemented as a table lookup using Read Only Memory (ROM). For 10 bit arithmetic, each ROM would require 1024 words or

10,240 bits. However, large wordlengths pose a problem. For example, 16 bit arithmetic requires 65,536 words or 1M bit, which is quite large! There is, however, a solution to this problem. Long wordlengths may be divided into a most significant half (MSH) and a least significant half (LSH). Each half can be multiplied separately, and the results combined with an adder. Figure 4 shows this double precision technique for 16 bit arithmetic, using only 6144 bits of ROM plus an adder. The ROM for the LSH is smaller than that for the MSH, since the LSH of the input only affects the LSH of the output.

There are numerous variations on the multiplier design just mentioned. For example, an alternative to bypassing the multiplications, as shown in Figure 2, is to multiply the data by 1 when $b_1=0$. This selection of the multiplier constant can be accomplished by using b_1 as an address bit for the tables, so that effectively a different set of tables is selected for $b_1=0$ and $b_1=1$.

This alternative to bypassing nicely lends itself to combining several of the multiplier stages in Figure 2 into a single stage by using the bits of the exponent for W as address bits for the tables. The result of this combination is to increase the table sizes while decreasing the number of adders and bypasses. Thus, when using commercially available components, excess amounts of "leftover" memory may be utilized in an efficient manner which reduces hardware. This combining of stages also helps reduce computational noise as mentioned in the next section.

Application of the above multiplier design to FFT pipelines yields additional advantages. Consider a radix 2 FFT pipeline for decimation in frequency [10]. At the first multiplier, all bits of the exponent for W are required. However, at the second multiplier the least significant bit is always zero. At the third multiplier, the least significant two bits are always zero, and so on down the pipeline. Thus, as we go down the pipeline, each multiplier requires one fewer stage than the previous multiplier, allowing a reduction in hardware. A radix 4 FFT pipeline is similar, dropping two stages in each successive multiplier.

III. COMPUTATION NOISE

A disadvantage of the table lookup multiplier scheme described in the previous section is that each stage within the multiplier (except the stages for multiplication by $-j$ and -1) introduces some computational noise. Thus, the computational noise is worse for this method than if a standard multiplier were used. It can be shown, however, that the additional noise is small.

As an example, consider a radix 2 pipeline using L bit integer arithmetic. It will be assumed that at the outputs of each butterfly, the data is divided by 2 to avoid overflows. Following an analysis similar to that in [2], it can be shown that for an FFT implementation using a standard multiplier the signal-to-noise ratio due to computational noise is given by

$$\text{SNR}_S \approx \frac{1}{7N} 2^{2L}. \quad (5)$$

A similar result can be derived for the table lookup implementation. If there are M guard bits internal to each stage of the table lookup multipliers, then for the table lookup scheme it can be shown that the signal-to-noise ratio due to computational noise is approximately

$$\text{SNR}_T \approx \frac{\frac{1}{12N} 2^{2L}}{1 + 3 \cdot 2^{-2M}} \quad (6)$$

where we have made use of the decreasing number of stages within the pipeline multipliers. For the case of 2 guard bits ($M=2$) the difference between eqs. (5) and (6) corresponds to 3.09 db.

Variations on this analysis are, of course, possible. For example, when several multiplier stages are combined into a single stage as mentioned in the previous section, there are fewer stages to introduce computational noise and SNR_T is thus closer in value to SNR_S . Similar results are obtained for radix 4 implementations except that SNR_S and SNR_T are greater than the radix 2 cases, due to a smaller number of butterflies.

In order to understand the effect of computational noise, it is necessary to know the ideal signal-to-noise ratio which would be obtained if infinite precision arithmetic were used. In many applications, the ideal SNR can be thought of as the SNR of the input data plus the processing gain. When the ideal SNR is significantly less than the SNR due to computational noise, the computational noise does not significantly affect the result.

Taking into consideration the ideal SNR, we show in Figure 5 a plot of the output SNR of the standard and table lookup multiplier implementations for a 512 point FFT using 16 bit arithmetic, with 2 guard bits ($M=2$) in the table lookup case. It is clear that, in this case, for an ideal SNR of 50 db or less the computational noise of either implementation is insignificant. For many practical applications, the ideal SNR is less than 50 db. Between 50 db and 60 db the difference in the two SNR's becomes apparent. Similar curves are obtained for other transform

lengths and wordlengths. For shorter transform lengths the breakpoint between the two curves occurs at a higher db level, and for shorter wordlengths the breakpoint occurs at a lower db level.

IV. SIZING

A preliminary sizing study has been done in order to compare the hardware required for the table lookup scheme with the hardware required in a standard multiplier implementation. In both cases, a pipelined radix 2 FFT is designed with off-the-shelf components. The amounts of hardware used by these designs is shown in Table 1 for transform lengths of 32 to 128 and wordlengths of 12 and 16 bits. In all cases, the table lookup approach uses slightly more hardware than the standard multiplier approach, but typically has a higher data rate.

In order to compare the various designs, we have defined a figure-of-merit as

$$\text{Merit} = \frac{\text{Transform Length} \times \text{Data Rate (MHz)}}{\text{IC Count}} \quad (7)$$

where the IC Count is the equivalent number of 16 pin integrated circuits used in the implementation. This definition of merit takes into account the ability of a design to process more data by using a longer transform, and to process more data in a given amount of time by using a higher data rate. This definition also takes into account the amount of hardware required so that a design using more hardware is given a lower merit. Wordlength is not included in the definition in eq. (7) since little is gained for excessively long wordlengths.

Figure 6 shows the figure-of-merit for the two implementations as a function of transform length. The two methods are seen to be very comparable in their performance.

These estimates were based on using commercially available components. However, if the designs were to use slightly customized integrated circuits of the same level of technology as those used in the designs discussed above, the hardware requirements of both the standard and table lookup techniques would be reduced. In particular, the table lookup method could benefit more than the standard method. This difference is illustrated by noting that, for example, the table lookup method uses more integrated circuits as data latches. These latches could easily be incorporated into the outputs of memories, adders, and other IC's. Table 2 shows the custom hardware requirements for the table lookup and standard implementations. In most cases the table lookup method requires less hardware than the standard method, which when combined with the higher data rate of the tables gives the table lookup method a distinct advantage. It is also believed that advances in memory technology and very large scale integration will further shift the balance in favor of the table lookup method.

V. CONCLUSIONS

In this paper, we have presented an efficient means by which table lookup multipliers may be used in the FFT. Preliminary sizing estimates indicate that by using off-the-shelf components, the table lookup implementation is comparable in performance to other FFT multiplier techniques. It is expected, however, that advances in integrated circuit technology will shift the balance towards smaller, faster, cheaper memories, making the table lookup multipliers very attractive. It is also expected that Very Large Scale Integration (VLSI) will allow the combination of many small components of the table multipliers into a single integrated circuit, making construction of table lookup based FFT structures an easier task than using existing components.

REFERENCES

- [1] Oppenheim, A.V., Applications of Digital Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1978.
- [2] Oppenheim, A.V. and Schafer, R.W., Digital Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1975.
- [3] Kolba, D.P. and Parks, T.W., "A prime factor FFT algorithm using high-speed convolution," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-25, pp. 281-294, August 1977.
- [4] Agarwal, R.C. and Cooley, J.W., "New algorithms for digital convolution," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-25, pp. 392-410, October 1977.
- [5] Winograd, S., "On computing the discrete Fourier transform," Mathematics of Comp., vol. 32, no. 141, pp. 175-199, January 1978.
- [6] Winograd, S., "Some bilinear forms whose multiplicative complexity depends on the field of constants," Mathematical Sys. Theory, vol. 10, pp. 169-180, 1977.
- [7] Agarwal, R.C. and Burrus, C.S., "Fast one-dimensional digital convolution by multidimensional techniques," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-22, no. 1, February 1974.
- [8] Liu, B. and Peled, A., "A new hardware realization of high-speed fast Fourier transforms," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-23, no. 6, December 1975.
- [9] Knuth, D.E., The Art of Computer Programming, Vol. 2 Seminumerical Algorithms, Addison-Wesley, Reading, MA, 1969.
- [10] Rabiner, L.R. and Gold, B., Theory and Application of Digital Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1975.

102319-N

PULSE-DOPPLER PROCESSOR

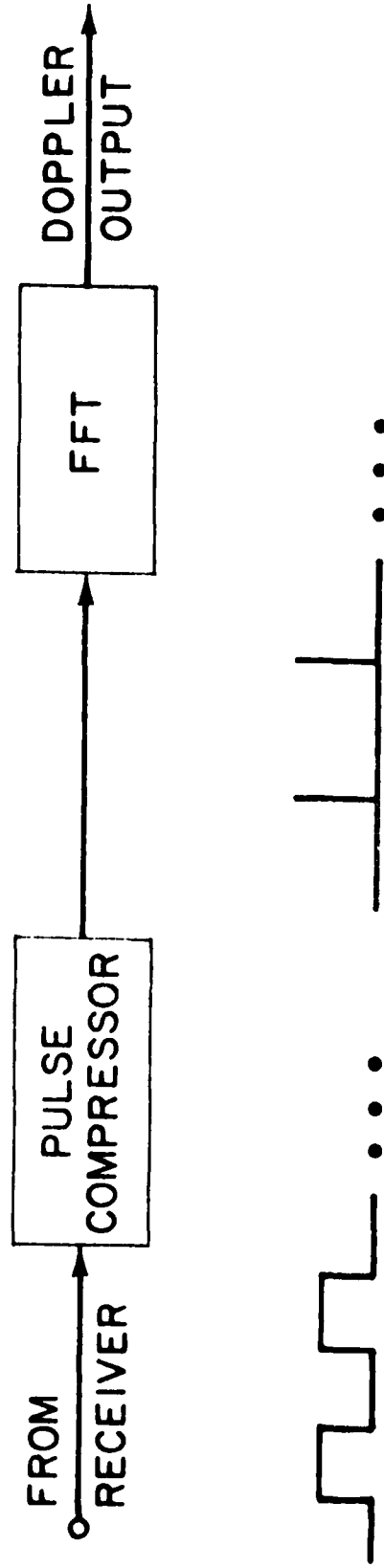


Figure 1. An application of the FFT to radar.

162321-N

CASCADE MULTIPLIER

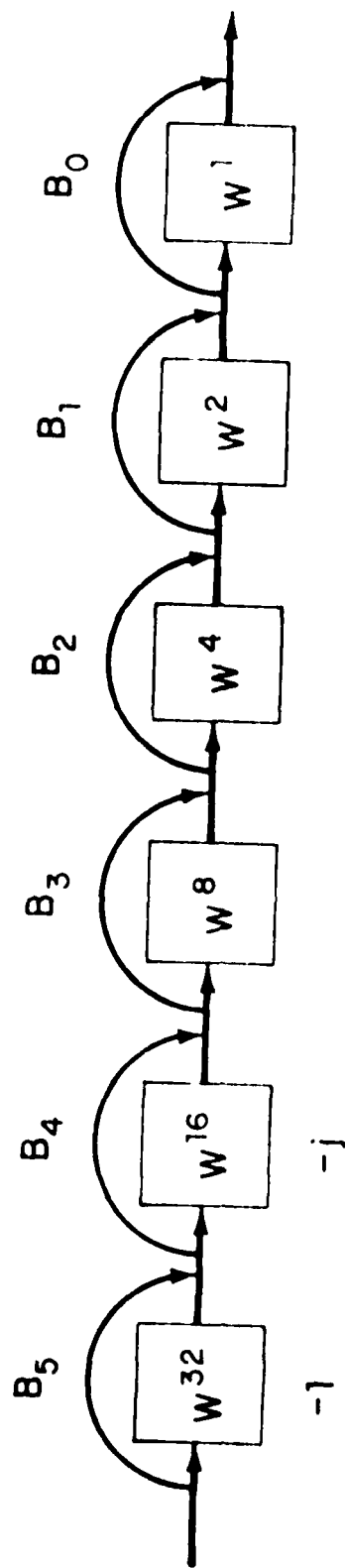


Figure 2. Multiplier for 64 point FFT.

102322-N

COMPLEX MULTIPLIER

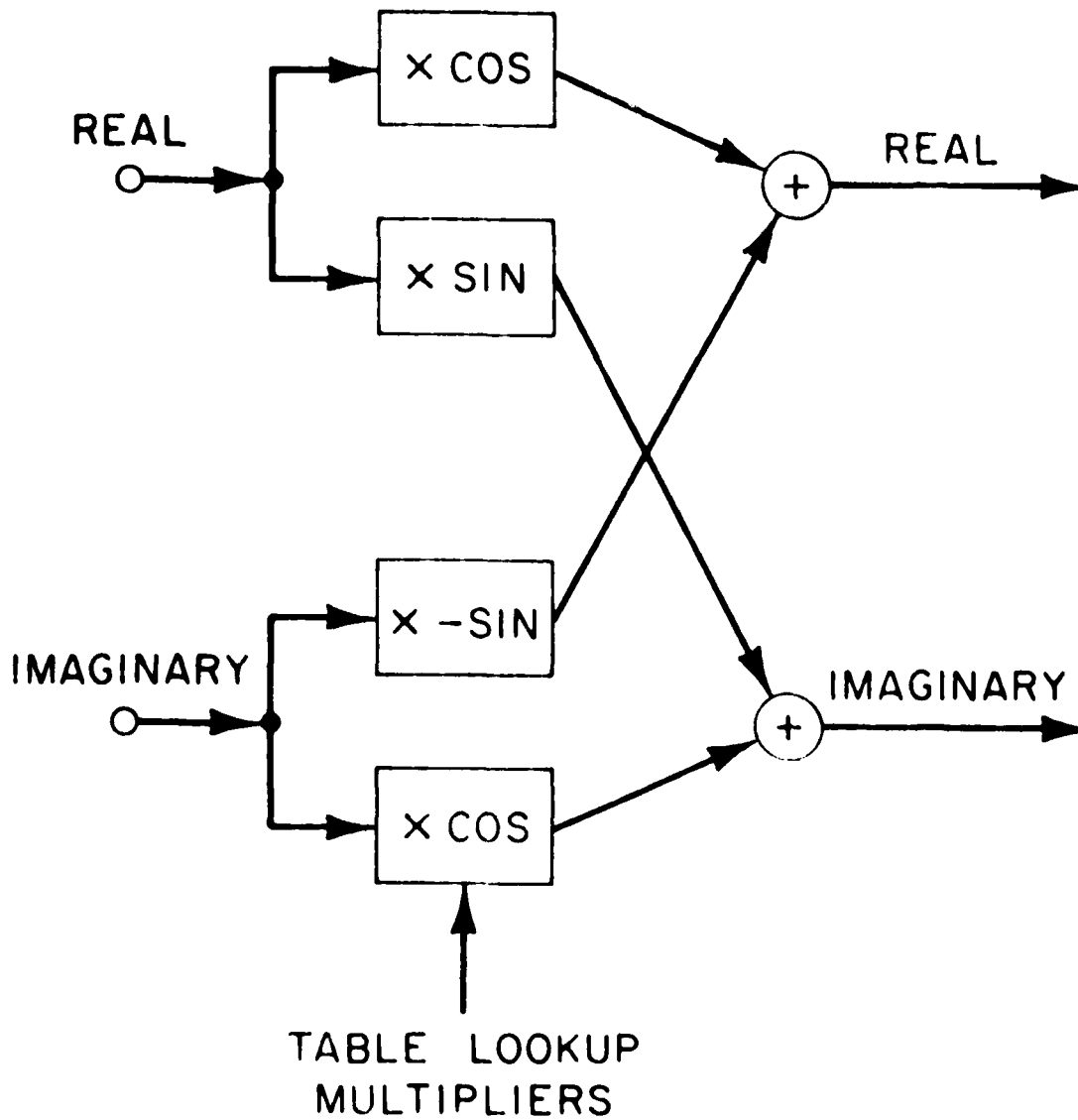


Figure 3. Single stage of the multiplier.

102327-N

DOUBLE PRECISION TECHNIQUE (16-bit words)

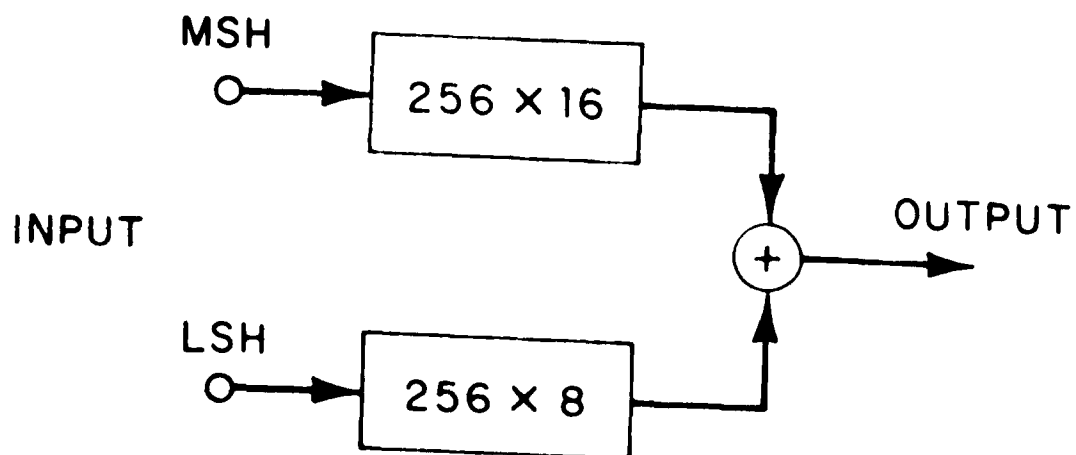


Figure 4.

DEGRADATION OF SNR DUE TO COMPUTATIONAL NOISE

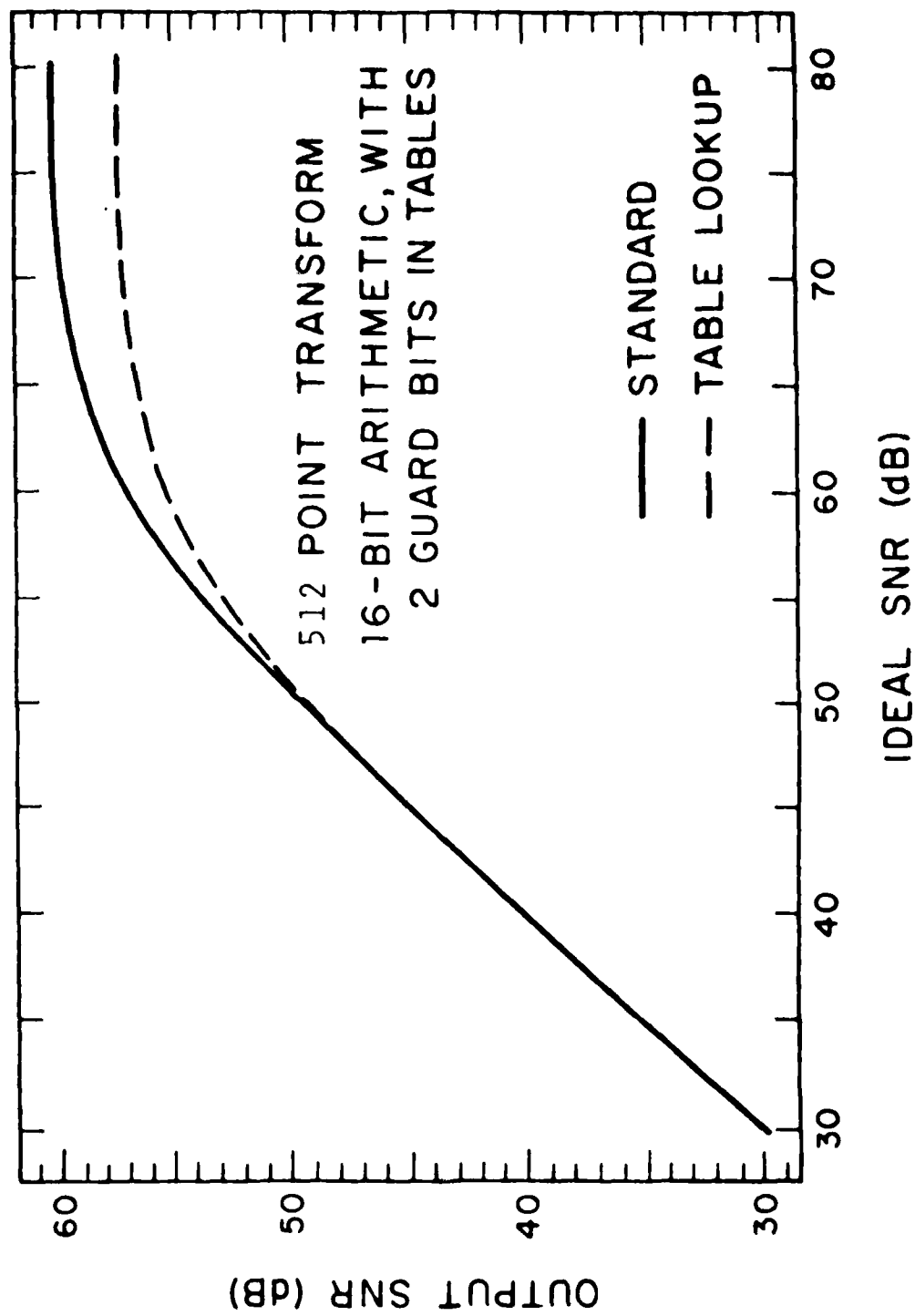


Figure 5. Computational noise effects.

SIZING

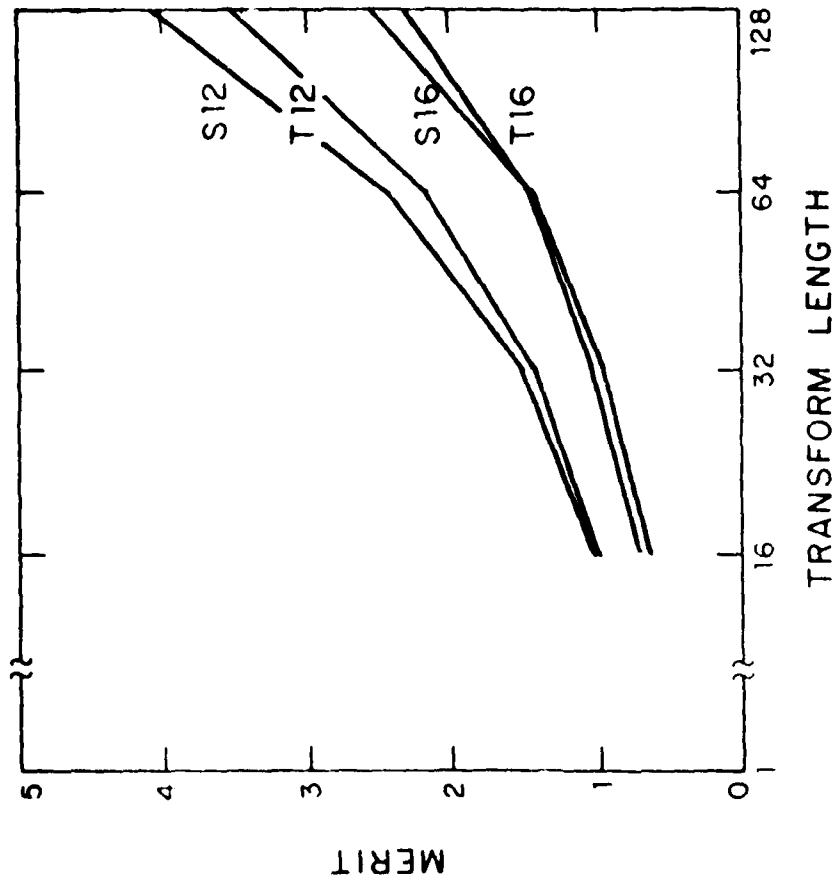
TRANSFORM LENGTH	TABLE LOOKUP		STANDARD	
	12-BIT WORDS (20 MHz)	16-BIT WORDS (20 MHz)	12-BIT WORDS (18 MHz)	16-BIT WORDS (14 MHz)
16	324	444	276	352
32	442	623	375	475
64	583	849	474	599
128	725	1105	574	724

HARDWARE REQUIREMENTS IN EQUIVALENT NUMBER
OF 16 PIN IC's

TABLE 1.

102325-N

FFT PIPELINES WITH OFF-THE-SHELF COMPONENTS



S = STANDARD, T = TABLE LOOKUP, SUFFIX = WORD LENGTH

$$\text{MERIT} = \frac{\text{TRANSFORM LENGTH} \times \text{DATA RATE}}{\text{IC COUNT}}$$

Figure 6. Comparison of the two techniques.

SIZING

TRANSFORM LENGTH	TABLE LOOKUP		STANDARD	
	12-BIT WORDS (20 MHz)	16-BIT WORDS (20 MHz)	12-BIT WORDS (18 MHz)	16-BIT WORDS (14 MHz)
16	183	256	207	258
32	242	353	278	344
64	320	450	349	430
128	396	571	419	516

HARDWARE REQUIREMENTS IN EQUIVALENT NUMBER
OF 16 PIN IC's FOR CUSTOM IC's.

TABLE 2.